I scratch and sense but can I program? An investigation of learning with a block based programming language

Journal Item

I Scratch and Sense but can I program? An Investigation of learning with a block based programming language.

N. K. Simpkins

The Open University
Department of Computing and Communications
Faculty of Mathematics, Computing and Technology
Walton Hall
Milton Keynes
MK7 6AA

Email: n.k.simpkins@open.ac.uk
Tel: +44(0)1908 858819, Fax: +44 (0)1908 652175

Neil Simpkins is currently a lecturer in web application development, in the department of computing and communications at The Open University. He has developed a range of short courses for the university's 'Web Application Development' certificate, as well as a major courses such as 'T320 E-business technologies: foundations and practice' and 'TT284 Web Technologies' which have been taken by many thousands of students. He has a doctorate in natural language processing and has worked as a technology expert in the European Commission and a number of major financial institutions as well as teaching in a range of other leading UK universities.

Abstract

This paper reports an investigation into undergraduate student experiences and views of a visual or 'blocks' based programming language and its environment. An additional and central aspect of this enquiry is to substantiate the perceived degree of transferability of programming skills learnt within the visual environment to a typical mainstream textual language.

Undergraduate students were given programming activities and examples covering four basic programming concepts based on the Sense programming language which is intended to simplify programming. Sense programming statements are represented by blocks which only fit together in ways that produce a meaningful syntactic outcome, which may lower the cognitive barrier to learning.

Students were also presented with concepts represented using an equivalent textual construct and asked to consider their understanding of these based on the graphical cases. They were finally asked to complete a short online survey. This paper presents the programming activities, the survey and an analysis of the results.

Keywords: skills transfer; visual programming; blocks language; technology education; distance education; Scratch programming.

1 Introduction

Teaching basic programming skills to novices has widely been seen as a problematic area responsible for the high dropout rates of around thirty to fifty percent (Dening & McGettrick, 2005; May & Dhillon, 2009) associated with computer science courses. Ma, Ferguson, Roper and Wood (2011) report that many first-year students perform worse than expected and that students can hold 'non-viable' mental models of programming concepts. They introduce a teaching model which utilises visualisation of programs as an aid to better understanding of key concepts. The visualisation they explore refers to visual models which depict the operation of a textual code fragment rather that the use of a blocks programming language itself to implement a program directly, which is the focus here.

When first encountered, basic programming concepts, such as repetition and conditionals, are inherently difficult to understand for a significant proportion of students. No single teaching approach seems to benefit all students. A proportion of students can readily comprehend a concept, such as repetition, as written in a textual language, for example Java, but a significant group will always be frustrated and require other forms of illustration of the concept which present less or differing cognitive load (Stachel et al., 2013).

This paper reports an investigation into the efficacy of a blocks based language for learning programming. The investigation has three stages for students:

1. Performing a series of simple programming activities using Sense as detailed in a brief guide.
2. Comparing the graphical programs that they have seen and run in step '1' with some textual equivalents which are similar to those found in more mainstream programming languages.
3. Completing a short online survey concerning their experiences.

In the activities students were provided with visual block programs to execute which demonstrated four basic concepts (sequencing, variables, repetition and conditionals) found in all programming languages. For two of these (repetition and conditionals) students were asked to consider if their understanding could be transferred to similar textual constructs. After completing these activities students were asked to complete a brief survey. Based on the responses the investigation focuses on two key questions:

1. How well a blocks programming language communicates basic programming concepts.
2. How easily an understanding of concepts based on a blocks programming language transfers to a mainstream textual language.

A student, having gained experience using a graphical language, will be faced with syntactic complexity and greater freedom for error when using a textual language. If central concepts themselves have already been learnt in a graphical environment then a student might be less burdened and better able to cope with greater syntactic and semantic freedom and complexity.

The initial parts of this paper cover background aspects, such as blocks programming languages, related work, the student population and the particular blocks language employed here. Subsequent parts examine the specific activities given to students and the response to the survey. The activities are available online (Simpkins, 2012) and the survey employed is given in Appendix A.

2 Background

Several innovations have sought to make programming more accessible by seeking to simplify aspects which are common sources of problems for a typical novice. Errors in syntax and semantic misuse of constructs are two prominent sources of problems.

Interactive Development Environments (IDEs) such as Eclipse (The Eclipse Foundation, 2013) provide context sensitive completion and help and also highlight syntactic errors in code. The syntactic and sematic complexity of languages such as Java means that these tools can be of limited assistance, if any, to novice programmers.

Blocks programming languages adopt a different approach. The programming language itself is made up of a range of objects, usually termed 'blocks', that can be dragged, dropped and plugged together to form a program. This approach to constructing a programme seeks to constrain the syntax of the language. Only blocks, which might be used together in a meaningful fashion, can actually be plugged together.

Lego Mindstorms (LEGO.com Mindstorms, n.d) is an example of a block programming language. This is designed for controlling a robot with each block representing a movement or other action such as firing a projectile. A program is constructed by dragging a sequence of blocks, each representing an action, out onto a work area to build up a program. According to its type each block can be configured using a range of parameters. So a movement block has parameters such as direction, power and duration. Mindstorms is not a general purpose language and is not as flexible as a language such as Java or Perl for example. It does illustrate how block programming languages can be innovative, accessible and quite powerful.

Other more recent developments of block languages have sought to introduce other features such as collaborative learning amongst students (Jain, Singhal & Gupta, 2011) and combine text with visual programming, such as Greenfoot (University of Kent in Canterbury, n.d.), which provides a visual depiction of the 'actors' (like 'sprites') and stage objects but which uses textual Java for programming.

3 Programming and the Open University

At the Open University (OU) teaching programming is a central area of interest within the faculty of Mathematics, Communication and Technology. The OU is the largest higher education institution in the United Kingdom with over 250,000 registered students. The OU only offers distance learning and has a long history of using technology for delivering courses and for supporting students online.

The demographics of the OU student population are quite different to those of a typical UK 'bricks and mortar' university. The average age of new undergraduate OU students is thirty-one and 9% of new students are over fifty.

Around 27% of new OU undergraduates were under twenty-five in 2012 (19,982 students) and over thirty-one thousand of all students were under twenty-five. In very recent times the makeup of student population has started to change. The general trend is toward greater numbers of younger undergraduates who in the face of increased university fees are seeking less costly alternatives. Distance learning with full or part-time options is one such solution. Of new students, 45% had one A-level or lower qualification. A high proportion of students (over 71%) also work full or part-time whilst studying.

4 Scratch visual programming

In 2009 the Open University started production of a new course 'TU100 My Digital Life' (Open University 2013; Richards, Petre & Bandara 2012). This course was important for the university, as it is the first course that new undergraduates in technology related fields must embark on and as a central entry level course it will be taken by thousands of students each presentation. Importantly, the course is the initial course for those starting computing related degrees and introduces students to programming concepts.

A previous entry level course teaching programming at the OU (Woodmn. Griffiths, Macgregor & Holland, 1999) had employed LearningWorks, a Smalltalk programming environment with graphical enhancements. Another course employed a traditional textual language (JavaScript) but received negative student feedback and significant numbers of students failed to complete the course. Based on this experience the team made the decision to adopt a blocks programming language, seeking to lessen the cognitive burden on students who are typically just embarking on academic study.

At that time 'Scratch' (Resnick et al., 2009; Lifelong Kindergarten Group n.d.) had started to emerge as a popular and engaging environment for young people to use. Scratch is aimed at pre-university education but includes all the basic elements of a general purpose programming language (repetition, conditionals, threads, etc.). Scratch does not have the facilities for more sophisticated notations such as inheritance or other types of abstraction. A more recent version of Scratch (version 2) does allow user-defined blocks which can be seen as equivalent to procedures (Lifelong Kindergarten Group, n.d.).

The Scratch environment is freely available, has a thriving user base, good documentation and there are example projects that can be downloaded. At the time of writing there are over three million such projects produced by users from around the world.

The graphical nature of the programming language precludes student errors in syntax. A program is created by first selecting a statement type such as 'Motion' which displays all statements in that category in a panel. A statement can then be dragged and dropped into the 'scripts' panel and can be slotted together with other blocks to create sequences of statements making up a program.

A range of example blocks available in Scratch is illustrated in Fig. 1, classified according to Malan and Leitner (2007). Blocks have tags, indentations and shapes so that they can only be plugged together in ways that make a syntactically meaningful statement. So for example hexagonal Boolean blocks can be inserted into a hexagonal hole in, for example, a conditional `if` block. Blocks which enclose other blocks, such as repetition and

conditional blocks dynamically resize themselves to enclose any number of other blocks as these are added or removed. The range of blocks available is quite extensive (Lifelong Kindergarten Group, (n.d.)).

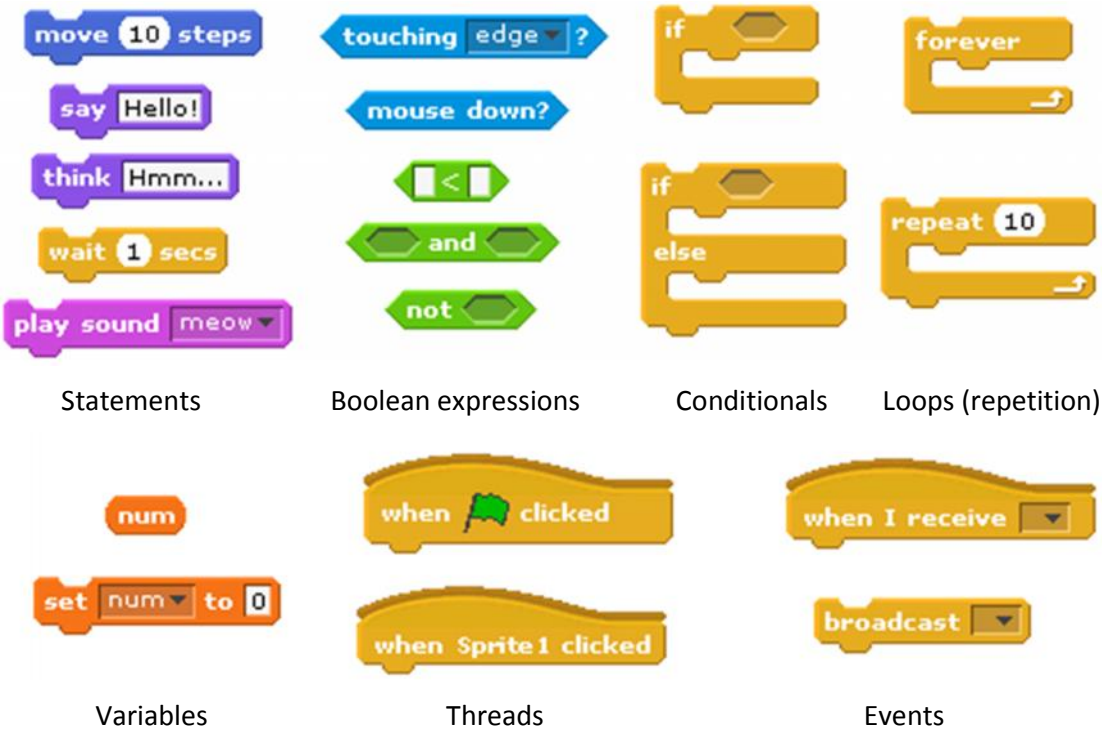


Fig. 1 Example Scratch blocks according to Malan and Leitners' classification.

Fig. 2 shows an example Scratch program. Here such a vertical sequence of blocks is termed a 'stack' of blocks. The stack executes from the block at the top of the stack and proceeds down the stack one block at a time. The program is started by clicking on a green flag button in the environment as indicated by the top block. Under this is a repetition block which runs the enclosed program blocks repeatedly 'forever'.

The first enclosed block causes the program to wait for ten seconds after which another second repetition block will execute the blocks it encloses ten times. The extent of the waiting period is controlled by simply typing a number into the white 'seconds' box, here ten, in the 'wait' block. This method of parameterisation is common to several other block types such as repetition for which the number of repetitions is entered.

Inside of the inner repetition block is a single block which moves the sprite on the stage by two steps. After twenty steps the sprite will 'bounce' if it has reached the edge of the stage area as specified by the last block in the forever block.

Fig. 2 Example scratch program stack of blocks

Not all blocks programming languages have such a facility but importantly Scratch has a 'stepping' mode where a program is executed one block at a time with the current position in the program being highlighted (Fig. 3). This allows students to follow the progress of execution of a program and to see which path is taken when there are choices, such as provided by a conditional statement.

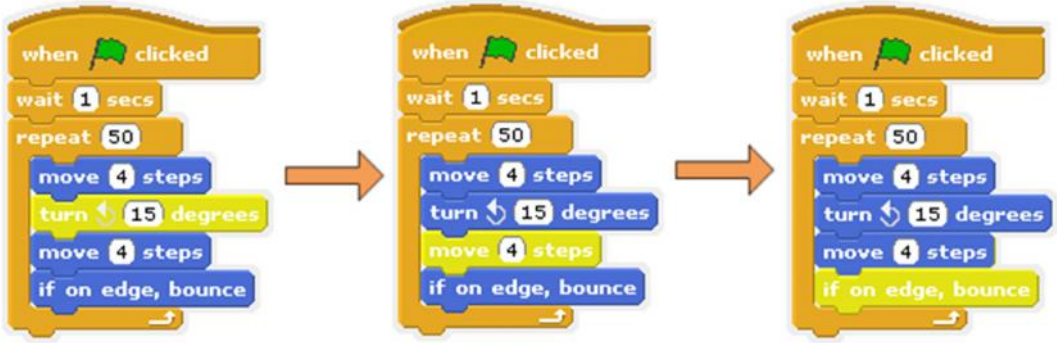


Fig. 3 Example of highlighting of blocks as execution progresses

Highlighting pauses when there is a 'wait' block or when there is interaction with the user, for example to gather input using a popup dialogue box. Students were asked to utilise this stepping facility when performing the programming activities that they undertook before taking the survey, so that they can more easily see the flow of control through each program stack.

There are now other variants of Scratch, such as Panther (Panther group, n.d.) which includes additional programming facilities but which is not intended to be used by novice programmers. Another variant Snap, (University of California at Berkeley, n.d.) adds other advanced features such as the ability to construct custom block types. It also supports higher order functions so that blocks may for example receive a function as if it were data and subsequently execute the function (Harvey & Mönig, n.d.).

Blocks languages may also have weaknesses for which there is some evidence but little literature. It is sometimes commented that they are inflexible and lack the expressive power of a mainstream textual language. This may be why new variants provide additional notional devices and there is no valid reason to believe that a blocks language might not be as powerful as any other programming language. There is also a question of screen space. As programs become more complex navigating multiple, perhaps large stacks, may become difficult. Newer visual devices such as shrinking procedural stacks to be shown as a single block may relieve this problem.

At the OU it was decided to use a modified version of Scratch, later named Sense, as the main tool for teaching programming in TU100. Sense, which is similar to PicoBoard (Playful invention company, n.d.), is briefly outlined in the next section.

5 Sense Visual Programming

Scratch was extended to produce the Sense environment (Fig. 4) (Richards et al., 2012). It extends the blocks language itself and adds a hardware board which is connected to a PC with a USB connector. The hardware board carries a range of devices to provide additional sensors and controls. The blocks language has been extended to support control and sampling of board components. The Scratch language is not changed in other significant ways and the interface resembles Scratch in general layout.

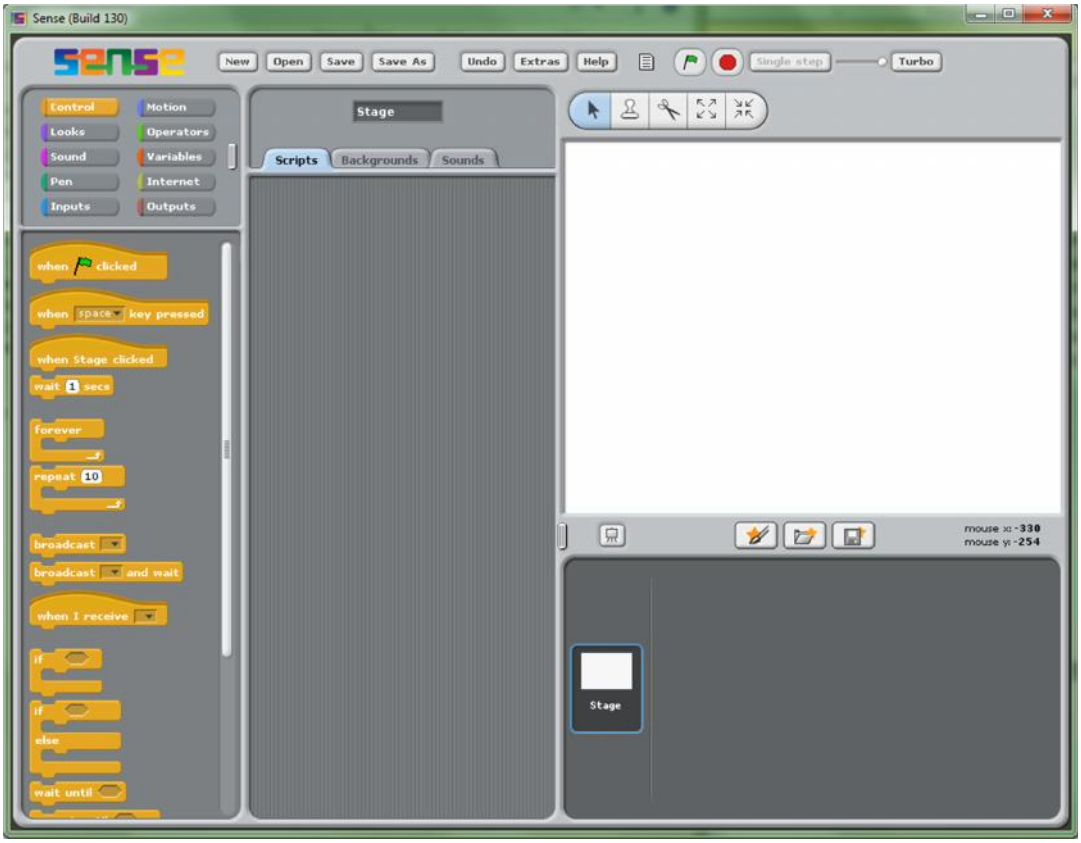


Fig. 4 Sense programming environment when started

The Sense board is not described further here because it was not utilised in the programming activities that were given to students. In fact the activities were produced in versions for Scratch and for Sense so that students not taking TU100 could participate. All of the students surveyed and reported here however have taken TU100 and many would have been familiar with Sense and the Scratch project on which it is based.

It is important to note that these TU100 students may not have engaged with the course's programming component. It is permitted to pass TU100 without submitting any of the programming assignments. In addition, the invitation to participate was sent after the conclusion of the course so as not to have any impact on student studies. The consequence is that the number of students who actually received the invitation is unknown, as is the number of respondents that actually failed TU100.

6 Related Research

Most of the literature around Scratch has been concerned with teaching of school children (Lamb & Johnson, 2011; Resnick et al., 2009) and examined the nature of groups using Scratch as well as the diverse range of projects they produced. This literature does not examine the effectiveness of the tool in teaching although it has observed that children find the approach absorbing and rewarding in terms of the facility to make interesting and entertaining projects.

Malan and Leitner (2007) proposed Scratch as a good first programming language. They argued that Scratch allows students to focus on an understanding of logic without also dealing with syntax. Of the twenty-five students in their investigation 76% reported that they considered Scratch to be a positive influence when they later came to learn Java. Students that did not feel positive or negative about the influence of Scratch (16%) were those that had prior programming experience. The investigation did not include aspects such as transferability or the graphical nature of the blocks language included here.

Parsons and Haden (2007) used Alice (Cooper, Dann & Pausch, 2000), a visual programming environment for developing three dimensional animations. The environment allows students to drag and drop graphical tiles to create a program. Tiles represent different types of statement which control the actions of characters in an animation (Fig. 3). Alice characters are similar to Scratch sprites, but have properties and methods similar to objects in a traditional object orientated language. The students in the study had prior programming experience and the investigation sought to determine to what extent students could transfer their skills onto the visual language. The conclusion was that "Alice's powerful graphical capabilities may actually be counterproductive when Alice is used as a teaching tool, rather than as an animation tool." Student feedback provided an indication that some students "may have become distracted by details of the animation process itself". As Scratch also supports animations of characters this problem might also be applicable to the tool.

Powers, Ecott and Hirshfield (2007) examined students' ability to transfer from Alice to C++. They report significant problems for students, chiefly that they did not understand the attention to syntax required for compilation and submitted work that included syntax errors. They also found that "Many students became discouraged when their programs did not compile and they concluded that they were inadequate programmers, even though they were able to program in Alice". Students also viewed Alice as an environment for children and concluded that what they had achieved was not "real programming". Again this could also be a problem common to Scratch.

Lewis (Lewis 2012) provides a comparison of student experiences using Scratch and Logo (Logo Foundation, 2013; Agalianos, Noss & Whitty, 2001). Sixth grade students (11-12 year olds) used Scratch in a classroom environment and after some directed teaching were asked questions concerning repetition and conditional statements expressed both as Scratch stacks and as Logo textual statements. Lewis found some evidence for repetition being better understood based on the Logo textual representation whereas conditionals were better understood using Scratch. It was hypothesised that textual representation might provide a lower level focus for attention which in some cases compensates for the lack of a visual representation.

Scaffidi and Chambers (2012) employed Scratch in a study to investigate the acquisition of programming skills over time. The study examined two key aspects. Firstly 'sophistication'

of projects in their use of different Scratch primitives in projects over time and secondly 'coding efficiency' as the time taken to develop a project. This was achieved by randomly selecting users from the Scratch web site, retrieving their projects and then analysing these for finesse and coding efficiency. Both measures were examined as trends over time by examining an individual project's history recorded as how many times it was saved and by comparing the use of primitives in more recent projects against earlier projects by the same author.

A downward trend was discovered in the demonstration of technical expertise, a level or downward trend was observed in efficiency and a high level of drop-out was observed. These trends were partially explained by a loss of the more expert programmers from the population. Other possibilities were considered to explain the trends, such as more expert programmers being less likely to explicitly demonstrate their skills and that the measures used are inappropriate.

7 Programming activities and illustrations

As students may not have engaged with Sense previously it is necessary to provide a basis on which students can base their learning and view of Sense. This background was established by providing a set of activities and illustrations of basic programming concepts.

To investigate how well a blocks language communicates basic programming concepts students were given a range of activities (Simpkins, 2012) covering four basic elements of programming, which can be found in the majority of mainstream languages:

1. Sequencing. This activity demonstrates how a program consists of a set of steps or instructions which are carried out one at a time in the order they are written. The Scratch stepping mode is itself a direct illustration of this principle which is used in the activity.
2. Variables. This activity illustrates how (global) variables are declared, named and store values such as numbers, text etc. which are used in a program.
3. Conditional statements. This activity employs an if conditional construct and an if..else.. (Fig. 1) construct providing a choice point between different sets of subsequent instructions.
4. Iteration or repetition. This activity uses a repeat repetition construct to allow a set of instructions to be performed multiple times; repeating them as many times as required.

For each element a simple activity or two has been created to demonstrate the concept using the blocks language of Sense. Students do not themselves have to do any programming as the demonstration programs are provided in a project form so they may be downloaded and simply opened in Sense.

It was estimated that each activity would not take more than a few minutes to complete, as would the survey. It was thought that students would not complete the exercise if it were too time consuming.

8 The Survey

The survey reported here was conducted based on the Sense environment because students had in principle completed TU100 and might have installed that version of the software. Students were invited on a voluntary basis to complete the survey after the end of the course, so as not to interrupt their studies. Whilst the activities were produced in two versions, tailored to Scratch and Sense, the survey itself is common to both and refers to the Scratch programming language and environment.

The survey was created using the LimeSurvey tool (LimeSurvey Project team, 2011). This tool is quite flexible and manages the storage and presentation of survey responses. It allows questions to be presented with each question as a separate web page with links to navigate backward and forward (Fig. 5). Two other options are offered on each web page so that a partially completed survey can be saved and resumed later or the survey can be abandoned.

All the questions required a mandatory answer except for the last which solicits any additional comments that a student might have. All the questions, except the last three, required students to select a single option as their response. The final question allows a free text response and the two before this allow students to select any number of the statements listed that they agree with.

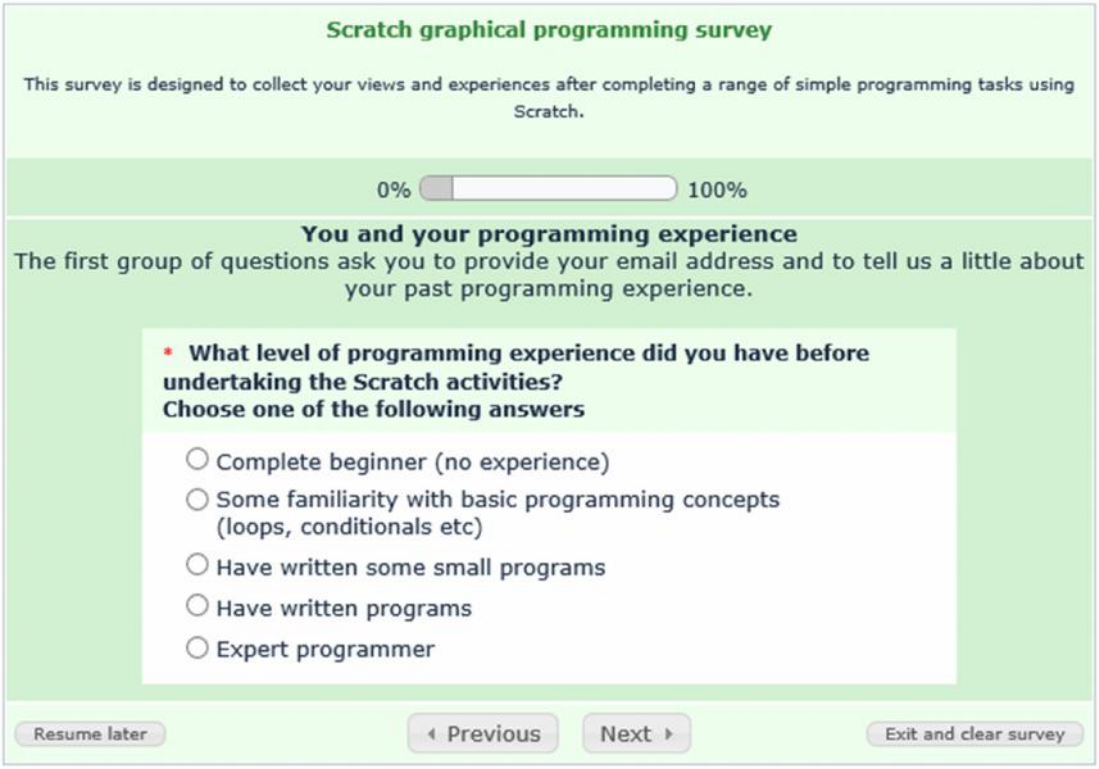


Fig. 5 LimeSurvey question web page

In the analysis of results that follows the questions have been shortened but the entire survey is listed in Appendix A.

A central feature of this enquiry is the degree to which students believe that the programming skills they have learnt within the visual environment can be transferred to a mainstream textual language. This aspect of the study is important, as learning how to

program in the blocks language is pedagogically not an end in itself. The Sense language is not as flexible or powerful as a mainstream language and is not suitable for, or utilised, in the IT industry. Using Sense may simplify learning of programming and is an entertaining approach but crucially, if it is to have real value, the skills learnt should be readily transferrable to other languages.

The survey itself investigates undergraduate students in terms of their:

- Experience of programming. Students' previous experience ranges from a complete beginner with no experience to expert programmers.
- View of the blocks programming language they have used across a few areas such as: ease of adoption, usability, flexibility, expressive power.
- Perception of the transferability of skills learnt using the blocks language to a typical textual programming language.

The format of the survey as a sequence of web pages should not itself influence the responses students selected (Downes-Le-Guin, Baker, Mechling & Erica, 2012) but does allow the survey to be more easily completed and is best suited to a highly geographically distributed cohort.

This survey is limited in scope in that it has not been possible to greatly investigate aspects such as 'positive bias' (Groves et al., 2004) where students might respond in a manner showing support to the blocks language when they think this is the desired outcome. The potential for positive bias has been reduced by the timing of the survey. Students were invited to participate after the conclusion of the course presentation and it was made clear that participation was voluntary and not associated directly with the course presentation. The brevity of the survey is intended to facilitate student participation and avoid so called 'survey fatigue' (Kampen, 2006), although it is likely that other factors have influenced the survey responses such as respondents avoiding selection of responses which are on the extremities.

9 Survey results

Only one invitation was issued to students to participate in the survey, to which there were seventy-five responses. Of these a very few are incomplete, as the latter questions have not been answered and the number of responses drops to a low of seventy-one for the last seven questions.

The survey is made up of sixteen questions. The first question simply asks for an email address to be entered. This is used to identify the student's answers if they decide they will suspend the survey to return later and complete the questions. In the following sections each of the subsequent questions is outlined and the responses examined.

9.1 Students' main area of educational interest

This question was intended to aid in profiling the participating students. It was something of a surprise (Fig. 6) to find so few students outside computing and technology but this is a Maths, Computing and Technology faculty introductory course.

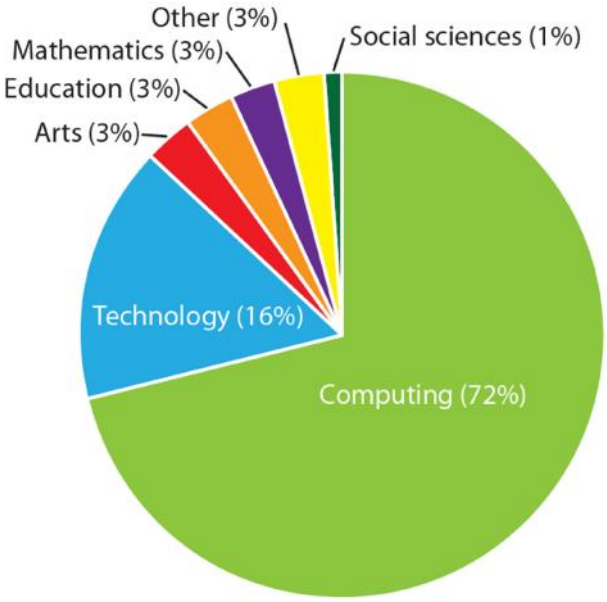


Fig. 6 Area of educational interest

Several areas of educational interest were not represented; languages, business, law and health. Typically an IT course in this subject area at the OU would attract a small number of students from the business school, so this was an unexpected finding. It might be that this minority, with less interest and feeling less connected to a future in IT at the university were not motivated to participate.

From the responses received it is to be expected that the students will be highly motivated in programming challenges as they view either computing or technology as their own chosen area of educational interest. However, this does not indicate that they will actually be well suited to programming.

9.2 Students' level of programming experience

The reported level of students experience in programming (Fig. 7) is a key element of this investigation. If most students were found to be experienced or expert programmers then their opinion of a blocks programming language as an introductory tool would be compromised, especially as it might well be that the majority of these students can be expected to have been taught previously using textual languages. Students were not asked if they had previous experience of a blocks or textual language to keep the survey short but it is thought unlikely that OU students embarking on a degree would have encountered a blocks language before.

Fig. 7 Level of programming experience

The majority of the respondents had little or no programming experience. It is not clear that the students putting themselves in the category 'Written programs' have excluded themselves from being something of a beginner in programming.

The 29% of students describing themselves as 'Complete beginner (no experience)' is unexpected as they have completed the TU100 course and they should therefore have performed a range of Sense programming activities. However, it is possible to pass TU100 without undertaking any of the Sense programming activities.

The 8% of students that have classified themselves as 'expert' may be seen as something of a contradiction given that TU100 is a first level introductory course. However, it is known at the OU (Open University, 2012) that most students are working (71%) and many have extensive experience as practitioners in industry but decide that their careers would benefit from a better technical foundation and by gaining certification from a recognised university.

As this investigation is focused on basic concepts and support for beginners expert programmers were asked to approach the activities and survey by reflecting back to when they started to learn programming. It must be expected that the experts would still have different views of the suitability of the block language and tool in some areas, especially in terms of their suitability for complex projects.

9.3 The sequencing activity

The first graphical illustration attempts to impart sequencing to students. This is a very simple concept and one that students do typically readily grasp even if they find other programming concepts difficult. The survey then asks two questions:

1. How well the activity as a whole illustrated the concept (Fig. 8).
2. How the graphical nature of the illustration supported understanding the concept (Fig. 9).

The overall satisfaction with the activity as illustrating sequencing is extremely high with 81% describing the concept as 'illustrated well'. This also demonstrates to some extent that the students were not subject to 'central tendency bias', where responses at the extremes are avoided, which might have been complicit in reducing this figure and

increasing the less extreme answers. The other two positive answers account for a further 15% of students, leaving just 3% reporting a negative view of the effectiveness with which the Sense program illustrated the basic concept.

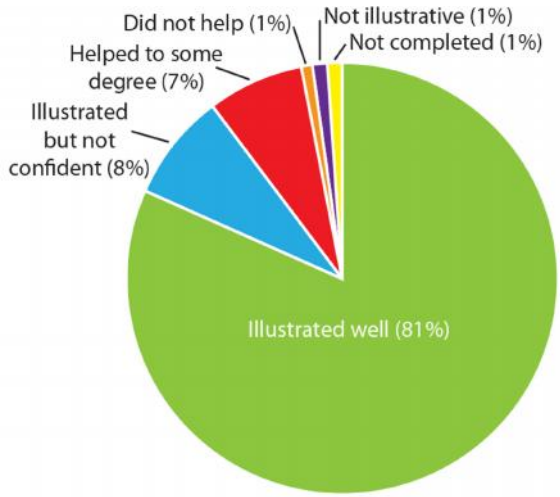


Fig. 8 Level to which sequencing activity illustrates the concept

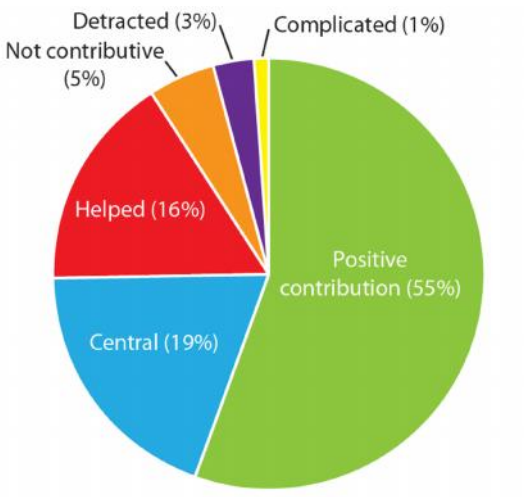


Fig. 9 Significance of the graphical nature of the sequencing illustration

Of the 8% that remain 'not confident' that they have understood the concept 50% classified themselves as complete beginners as programmers. The very high level of satisfaction with the illustration is somewhat surprising and perhaps a testament to the power of a graphical approach.

The second question of this pair resulted in the response depicted in Fig. 9. In this question the 'central' group represent the most positive response and 'positive contribution' the next lesser positive. The three positive answers were selected by a total of 90% of students. This very high positive response demonstrates a very strong feeling that it is the graphical nature of the illustration which provides students with insight to understand the concept.

The very high levels of positive statements in response to both questions are very supportive of the Sense blocks language as a tool for learning.

9.4 The use of variables activity

The next two questions relate to the activity which outlines how variables can be used to hold values and how these can be manipulated in operations such as addition.

Again students are asked two questions concerning 'using variables' and the graphical program's illustration:

1. How well the activity as a whole illustrated the concept (Fig. 10).
2. How the graphical nature of the illustration supported understanding the concept (Fig. 11).

The graphical program as an illustration of the concept is slightly less positive (Fig. 10) in terms of the most positive answer (71%) but the three positive answers together represent 95% of student responses.

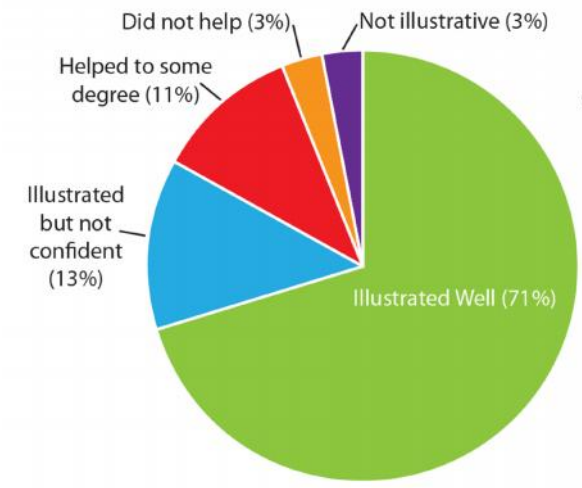


Fig. 10 Level to which the 'using variables' activity illustrates the concept

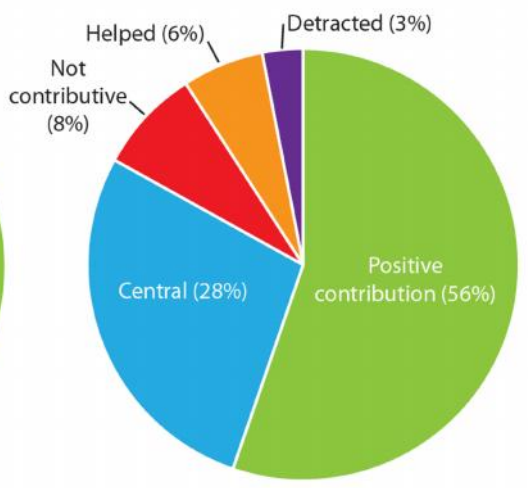


Fig. 11 Significance of the graphical nature of the sequencing illustration

The contribution from the graphical nature of the illustration to understanding of the concept is reported to be greater than in the previous case with both 'central' and 'positive contribution' being increased.

Again it is a surprise that such a very high percentage of students believe they have understood the concept and that they cite the graphical nature of the illustration is the foundation of this.

9.5 The repetition activity

The third programming concept is that of repetition. This concept and that of conditionals, the last concept, are slightly more complex and thought more difficult to comprehend for novice programmers. It was therefore expected that fewer students would understand the illustration of the concepts in these cases.

The response to how well the activity illustrated the repetition concept is actually better with 80% returning the most positive response and a total of 94% selecting the three positive responses. So despite the increased complexity of the concept it appears that students are more supported by the illustration. This might be because they feel that the insight is of greater value.

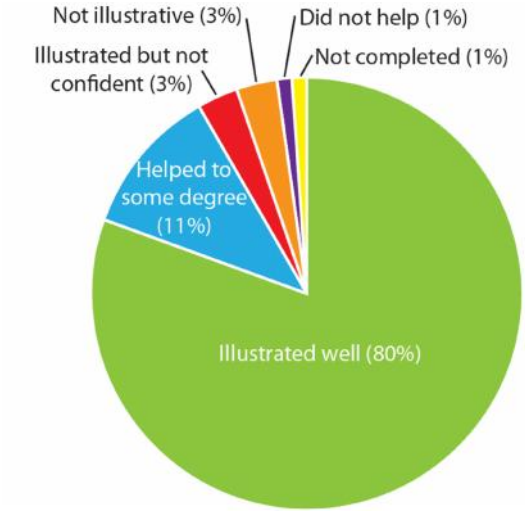


Fig. 12 Level to which the repetition statement activity illustrates the concept

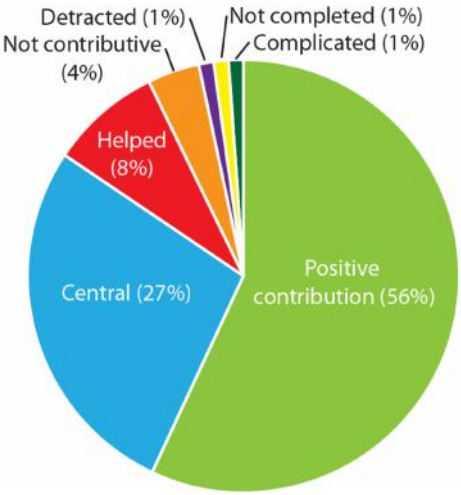


Fig. 13 Significance of the graphical nature of the repetition illustration

The response specifically to the graphical nature of the illustration is very close to that of the previous activity. The overwhelming view is consistent in that the graphical nature appears very supportive to understanding.

9.6 The conditional activity

The last programming concept is that of conditional statements. An 'If <condition> then <action1> else <action2>' statement was employed to illustrate this. The condition tests some input provided by a pop up dialogue box into which the user types a response. Each action consists of a pop up message dialogue box reporting which action is being executed. The dialogues allow the user to select which of the actions is performed (the 'then' action or the 'else' action) and to receive clear feedback concerning which action is performed.

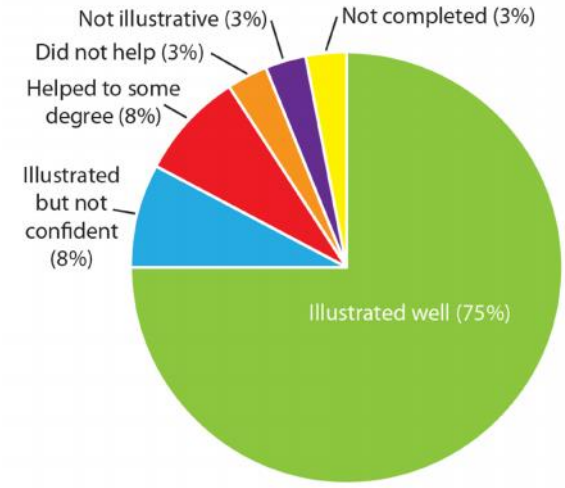


Fig. 14 Level to which the conditional statement activity illustrates the concept

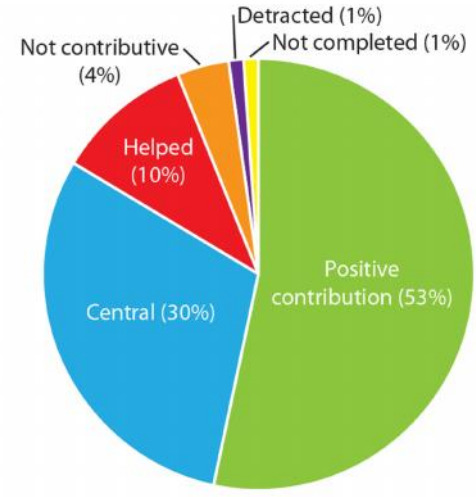


Fig. 15 Significance of the graphical nature of the conditional illustration

The illustration of the conditional statement (Fig. 14) shows slightly less success than the illustration of iteration. Whilst 75% of students thought the concept was illustrated well there are significantly more that are less confident they have actually understood the concept, despite considering the illustration to be satisfactory.

Whilst fewer students have understood the concept of a conditional statement the contribution of the graphical nature of the illustration is very much the same as for the earlier iteration. Again it appears to be precisely the graphical nature of the illustration that has proven useful for students in understanding.

10 Transference to textual constructs

Two of the survey questions concerned the ability of students to understand textual programming statements based on their familiarity with a graphical equivalent as encountered earlier during the programming activities. The questions concern the ease of understanding a repetition and a conditional. For each of these, students were provided with the Sense code as a stack of blocks and a textual equivalent to the stack written in a fictitious language.

10.1 Conditional

The conditional statement used as an illustration has the form of an 'If <condition> Then <statements> Else <statements>'. The Sense stack of blocks is shown in Fig. 16. This is again a program that starts to run when the green flag button in the Sense interface is clicked. The second block is a 'set variable to value' block which here sets the value of the selected variable called 'answer' to a value returned by another embedded block which is a 'pop up and get input block'. As the name implies this block produces a pop-up dialogue box and then returns whatever the user types into this when the dialogue's 'ok' button is pressed. The dialogue box will contain the pop-up's text as a prompt which here is 'shall we execute the blocks?'.

The third block is a 'pop up warning block' which will echo back to the user what they typed in, which is expected to be 'yes' or 'no'. The next block is the conditional 'if' statement block. If the user has typed in 'yes' then the pop-up warning block will inform the user 'Doing the IF part as you wished.' otherwise the next pop-up will inform the user 'Doing the ELSE part as you wished.'. A final block informs the user that the program has finished.

Fig. 16 Sense conditional statement stack

This Sense stack provides a very simple and interactive graphical illustration of how a conditional 'if' statement operates. The user also has direct control over which of the conditional paths is executed and receives clear feedback.

The textual equivalent of the Sense stack attempts to be as similar as possible to the Sense stack. As students may not be familiar with any textual language the equivalence between blocks and textual statements is briefly outlined in the activity guide:

"Here we are going to use:

- read (answer) in place of the 'get input' part of the 'pop up and get input' block, which puts a value into 'answer'.
- write(<something>) in place of a 'pop up warning' block which displays either some text or the value in a variable.
- instead of a 'appended with' operation on strings, we simple use '+' to join two string together (for example "Hello " + "there" produces a string "Hello there").
- a semi-colon ';' which marks the end of a statement. This is just a syntactic convention of the type of textual language we are thinking of."

The textual program code which does differ from the graphical illustration but was given as roughly equivalent is:

```
write("shall we execute the blocks?");
read( answer );
write( " you typed " + answer );
if ( answer = "yes" )
{
      write( "doing the if statement's IF part" );
}
else
{
      write( " doing the if statement's ELSE part" );
};
write('Finished.');
```

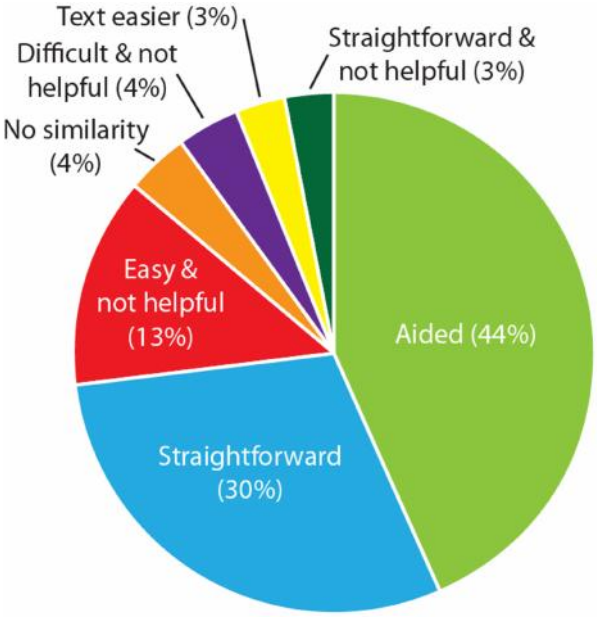


Fig. 17 Extent to which the graphical conditional supported understanding of the conditional textual construct

The support provided by the graphical conditional (Fig. 17) for understanding the textual version is less extensive than the very high approval of the graphical illustration seen earlier (Fig. 15). The highest level of support labelled 'Straightforward' was selected by only 30% of students but the total of the two answers which indicate the graphical approach made a positive contribution to understanding total 74%.

Perhaps the most interesting group are those that did not find the graphical illustration helped them which total 20%. However of these the greater part (13%) were those that nevertheless did find the textual conditional easy to understand.

10.2 Textual repetition

The Sense repetition block is included in a very simple set of blocks (Fig. 18). The program uses a variable called 'times to do loop' as a counter which specifies how many times to repeat the enclosed blocks. The value of the variable is obtained before the repetition is entered by use of a pop-up dialogue which asks the student to enter a numerical value for this purpose.

Inside of the repetition a 'wait' block is used to ensure the repetition executes slowly and that progress can be easily seen in the stepping mode which highlights each block as it executes. In addition, another variable 'times done' is incremented each time the repetition is executed. The value of this variable, as all others in the stack, will be displayed in the Sense interface. This incrementing value is intended to illustrate how repetition execution progresses from zero iterations toward the 'times to do loop' number of iterations.

Fig. 18 Sense 'repeat' block in a program stack

The activity text explains assignment and a repetition so that the textual version of the Sense program can be understood:

> "To set a variable called 'number' to the value '10' we will use a textual statement:
>
> number := 10;
>
> The 'wait' blocks in the stack we will simply ignore, although textual languages do sometimes have equivalent statements. The 'repeat <answer>' block is replaced by a 'repeat <answer> times' textual statement."

The textual repetition program was given as:

```
times_done := 0;
write("How many times shall we loop?");
read( answer );
repeat answer times
{
      times_done := times_done + 1;
}
```

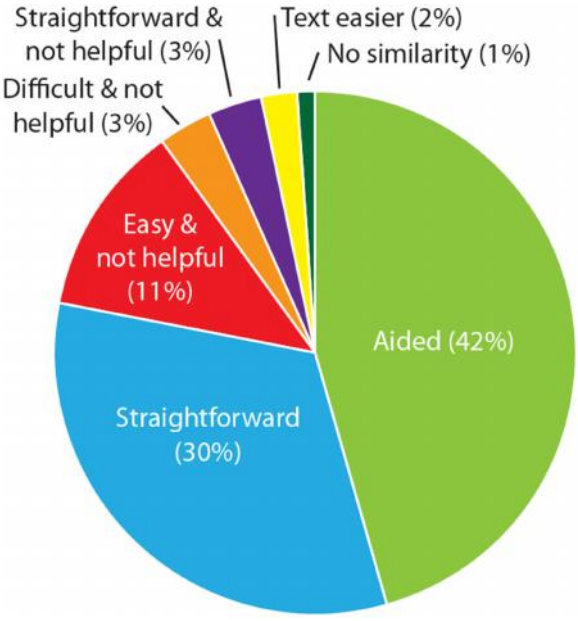


Fig. 19 Extent to which the graphical conditional supported understanding of the textual repetition construct

The textual repetition seems to be slightly less well understood than the conditional (Fig. 19). Again the two answers that indicate that the graphical illustration did contribute to understanding have a significant total of 72% but fewer students felt the construct was ‘straightforward’ to understand based on the graphical illustration. This small decrease is insignificant.

11 The Graphical Tool and Approach

Two further questions asked students about the Sense tool itself and the graphical approach to programming that Scratch provides. For these questions students were allowed to select as many options as they agreed with. These questions are also broader in scope and designed to identify factors or lessons that could be learnt from the use of the Sense blocks language and the Sense graphical environment.

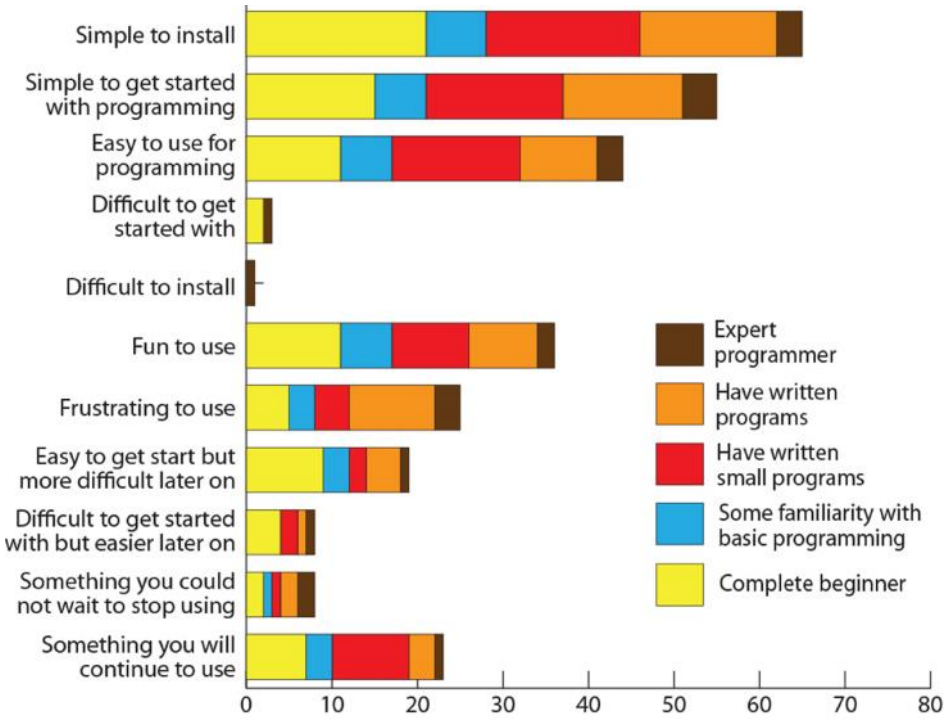


Fig. 20 The Scratch/Sense tool

The most popular options (Fig. 20) strongly suggest that the software is easy to install, start using and subsequently to use for programming. The degree to which students considered the software ‘fun to use’ and the number which report they will continue to use the software after the end of the TU100 course is surprisingly high.

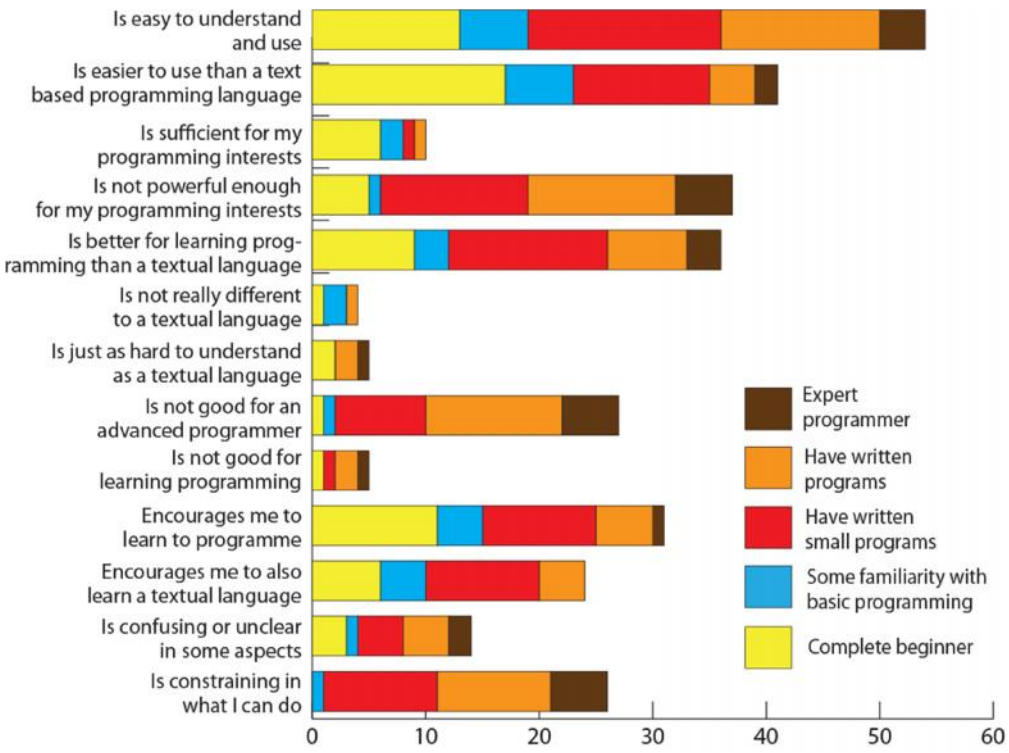


Fig. 21 The graphical approach to programming

The options selected which relate to the graphical approach to programming (Fig. 21) clearly indicate that, as expected the approach is easier to understand and easier to use than a textual based language and is especially appealing to those with little or no programming experience. However, there is a significant recognition that the Sense blocks language is limited in power and will eventually be constraining. The extent to which this is realised is something of a surprise. As expected, this is less recognised by complete beginners but is reported by those with significant experience.

12 Future Work

The investigation reported here first provided students with some programming activities to undertake so that they would gain a common and specific experience of Sense and then garners their opinions of the Sense tool, its blocks language and the transfer of skills to a textual language. In the future it will be possible to extend this work across a range of fronts.

One area for focus may be the testing of students to determine if their perception of what they have learnt is indeed correct. Both the understanding of Sense stacks and textual language might be assessed. Whilst OU students are mature students who will reflect and consider their responses to a survey, such as that employed here, it may be the case that they have been over optimistic in their responses.

Another area for further investigation would be to increase the complexity of the program examples. This might include higher concepts such as abstraction and nested constructs. It would also be of interest to re-survey students after they have learnt a textual language to assess if the perceived advantages of Sense remain in the longer term

given greater experience and if students did continue to use Sense as they intended. To do this will require some organisational approval to contact those from the TU100 cohort as they continue studying. A future survey could also examine in detail the reasons for the reduction in the numbers of students that understood the textual versions of constructs compared to the very high numbers that thought they had earlier understood the basic concept behind the graphical and textual constructs.

To facilitate these developments will generally require securing a greater commitment from a set of students than is normally possible at the OU. There is however a procedure and ethical means to achieve this which should allow the research to continue in these directions.

13 Conclusions and Summary

The survey results illustrate that both a complete beginner and those with a small amount of experience of programming believe they benefit very greatly from utilising a blocks language. The use of a blocks language is seen by students, with a very high degree of consensus, as very beneficial to their insight into basic programming concepts and it is the graphical nature itself of the language which supports this understanding.

Students reported that the blocks language significantly enables understanding of the most basic concepts (variables, stepping execution) as it does slightly more complex but still core concepts such as conditionals and repetition.

The overall conclusion is that students see the Sense blocks programming language as a powerful approach to adopt for supporting students in their early experiences of programming. Sense and Scratch are also reported to be simple to install, get started with and fun to use more generally.

Importantly, students report that they consider the transferability of the programming skills they have learnt in a blocks environment to a traditional textual language as very straightforward. At the same time a significant proportion of students consider the language and tool will ultimately be limiting in what they can produce.

The positive findings from this survey are in line with what might be generally expected but the degree to which students find the blocks programming language simple and easy to understand and to which they consider skills learnt can be transferred very greatly surpasses expectations.

Acknowledgements

I would like to acknowledge the support of the OU in producing this article and to thank Mr John Busvine of the University for his Support in conducting the survey reported here and Dr Karen Kear for her input.

References

Agalianos, A., Noss, R. & Whitty, G. (2001). Logo in Mainstream Schools: The Struggle over the Soul of an Educational Innovation. *British Journal of Sociology of Education*, 22(4), 479-500.

Denning, P. J., & McGettrick, A. (2005). Recentering computer science, *Communications ACM*, 48(11), 15–19.

Downes-Le-Guin, T., Baker, R., Mechling, J. and Erica, R. (2012). Myths and realities of respondent engagement in online surveys, *International Journal of Market Research*, 54(5), 613-633.

Eclipse Foundation. 2013. *The Eclipse Foundation open source community website*. Retrieved October 28, 2013, from http://www.eclipse.org/.

Groves, R., Fowler, F., Couper, M., Lepkowski, J., Singer, E. & Tourangeau, R. (2004). *Survey methodology*, Hoboken, NJ: Wiley & Sons.

Harvey, B. & Mönig, J. (n.d.). *SNAP! Reference Manual 4.0*. Retrieved October 28, 2013 from http://byob.berkeley.edu/SnapManual.pdf.

Jain, A. K., Singhal, M. & Gupta, M. S. (2011*).* Algorithm building and learning programming languages using a new educational paradigm, *AIP Conference Proceedings*, 1373, 149-158.

Kampen, J. K. (2006). The impact of survey methodology and context on central tendency, nonresponse and associations of subjective indicators of government performance. *Quality & quantity*, 41, 793-813.

Lamb, A. & Johnson, L. (2011). Scratch: Computer Programming for 21st Century Learners. *Teacher Librarian*, 38(4), 64-68.

LEGO. (n.d.). *LEGO.com Mindstorms*, Retrieved October 28, 2013 from http://mindstorms.lego.com/en-us/Default.aspx.

Lewis, C. M. (2010) How programming environment shapes perception, learning and goals: logo vs. scratch. *Proceedings 41st ACM technical symposium on Computer science education*, 346-350.

Lifelong Kindergarten Group, (n.d.). *Scratch: a programming language for everyone*, Retrieved October 28, 2013 from http://scratch.mit.edu/.

Lifelong Kindergarten Group, (n.d.). *Scratch 1.4 Reference Guide*, Retrieved October 28, 2013 from http://info.scratch.mit.edu/support/reference_guide_1.4.

Lifelong Kindergarten Group, (n.d.). *Scratch 2*, Retrieved October 28, 2013 from http://wiki.scratch.mit.edu/wiki/Scratch_2.0.

LimeSurvey Project team. (2011). *LimeSurvey - the free and open source survey software tool !*, Retrieved October 28, 2013 from http://www.limesurvey.org/.

Logo Foundation. (2013). *The Logo Foundation*, Retrieved October 28, 2013 from http://el.media.mit.edu/logo-foundation/.

Ma, L., Ferguson, J., Roper, M. & Wood, M. (2011). Investigating and improving the models of programming concepts held by novice programmers. *Computer Science Education,* 21(1), 57-80.

Malan, D. J. & Leitner H. H. (2007), *Scratch for budding computer scientists*, Procs. 38th SIGCSE technical symposium on Computer science education, 223-227.

May, J. & Dhillon, G. (2009). Interpreting Beyond Syntactics: A Semiotic Learning Model for Computer Programming Languages. *Journal of Information Systems Education*, 20(4), 431-438.

Open University. (2012). *The Open University in facts and figures*, Retrieved October 28, 2013 from http://www.open.ac.uk/about/main/the-ou-explained/facts-and-figures.

Open University. (2013). *TU100 My Digital Life*, Retrieved October 28, 2013 from http://www3.open.ac.uk/study/undergraduate/course/tu100.htm.

Panther group. (n.d.). *Panther - based on Scratch*. Retrieved October 28, 2013 from http://pantherprogramming.weebly.com/.

Parsons, D. & Haden, P. (2007). Programming osmosis: knowledge transfer from imperative to visual programming environments, *Proceedings 20th Annual Conference of the National Advisory Committee on Computing Qualifications*, 209-215.

Playful invention company. (n.d.). *PicoBoard*. Retrieved October 28, 2013 from http://www.picocricket.com/picoboard.html.

Powers, K., Ecott, S. & Hirshfield, L. (2007). Through the looking glass: teaching CS0 with Alice, *Proceedings of the 38th SIGCSE technical symposium on computer science education*, 39(1), 213-217.

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B. & Kafai, Y. (2009). Scratch: Programming for All, *Communications of the ACM*. Nov. 52(11), 60-67.

Richards, M., Petre, M. & Bandara, A. (2012). Starting with Ubicomp: using the SenseBoard to introduce computing, *In: 43rd ACM Technical Symposium on Computer Science Education*, 29 February-3 March 2012, Raleigh, NC.

Scaffidia, C. & Chambersa, C. (2012). Skill Progression Demonstrated by Users in the Scratch Animation Environment, *International Journal of Human-Computer Interaction*, 28(6), 383-398.

Simpkins, N. K. (2012). *Sense survey activities*, Retrieved October 28, 2013 from http://norton.open.ac.uk/senseStepsGuide.html.

Stachel, J., Marghitu, D., Brahima, T. B., Sims, R., Larry Reynolds, L. & Czelusniak, V. (2013). Managing Cognitive Load in Introductory Programming Courses: A Cognitive Aware Scaffolding Tool. *Journal of Integrated Design and Process Science,* 17(1), 37-54.

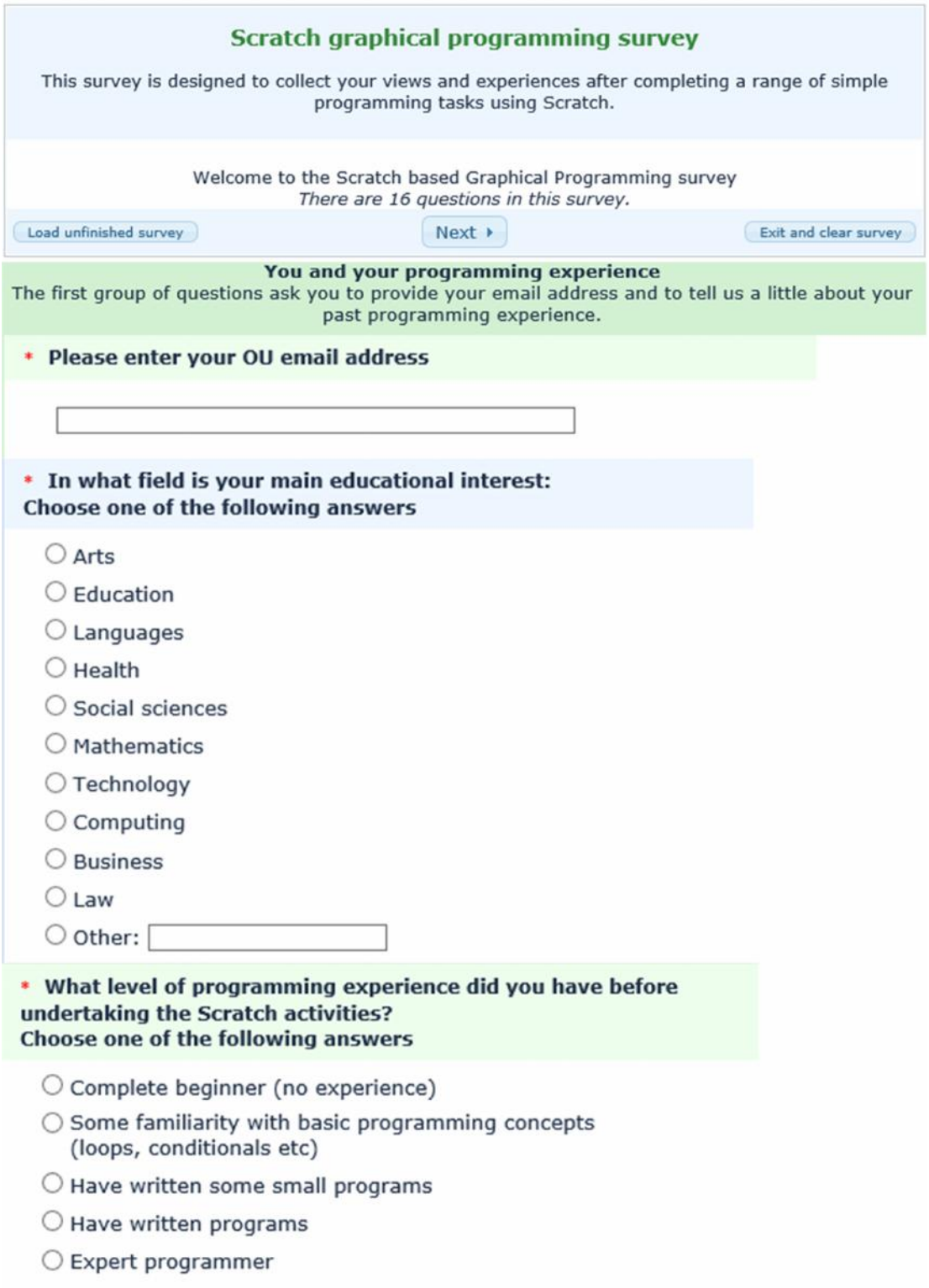
University of California at Berkeley. (n.d.), *SNAP! (Build Your Own Blocks) 4.0*. Retrieved October 28, 2013 from http://byob.berkeley.edu/.

University of Kent in Canterbury. (n.d.), About Greenfoot. Retrieved October 28, 2013 from http://www.greenfoot.org/overview.

Woodman, M., Griffiths, R., Macgregor, M., Holland, S. (1999), LearningWorks: A Customized Programming Environment for Smalltalk Modules. *Proceedings 21st International Conference on Software Engineering*, 638-641.

Appendix A Survey Questions

Here only the actual questions have been reproduced out of each web page that presents the question to decrease the size of the graphics. The questions are presented in the same order as they are presented to students completing the survey.



Scratch graphical programming survey

This survey is designed to collect your views and experiences after completing a range of simple programming tasks using Scratch.

Welcome to the Scratch based Graphical Programming survey
There are 16 questions in this survey.

Load unfinished survey | Next ▸ | Exit and clear survey

You and your programming experience
The first group of questions ask you to provide your email address and to tell us a little about your past programming experience.

* **Please enter your OU email address**

* **In what field is your main educational interest:**
Choose one of the following answers

- ○ Arts
- ○ Education
- ○ Languages
- ○ Health
- ○ Social sciences
- ○ Mathematics
- ○ Technology
- ○ Computing
- ○ Business
- ○ Law
- ○ Other:

* **What level of programming experience did you have before undertaking the Scratch activities?**
Choose one of the following answers

- ○ Complete beginner (no experience)
- ○ Some familiarity with basic programming concepts (loops, conditionals etc)
- ○ Have written some small programs
- ○ Have written programs
- ○ Expert programmer

The Graphical Activities

This second group of questions will ask you about your experiences of the graphical programming activities. If you are an experienced programmer, please try to answer the questions by looking back to when you were first learning to program. There are two questions about each of the four activities.

* **The 'stepping through a program' activity is intended to illustrate how a program is executed – one instruction after another. Do you think this activity:**
Choose one of the following answers

- ○ Illustrated the concept well and I am confident that I understand the concept
- ○ Illustrated the concept well but I am not confident that I understand the concept fully
- ○ Illustrated the concept to some degree and this helped me somewhat to understand the concept
- ○ Illustrated the concept to some degree but this did not help my understanding
- ○ Did not really illustrate the concept
- ○ I did not complete this activity

* **The graphical nature of the 'stepping through a program' activity:**
Choose one of the following answers

- ○ Was central to the illustration and the understanding to be gained from the activity
- ○ Contributed in a positive fashion to the illustration and my understanding
- ○ Helped in my gaining an understanding
- ○ Did not really contribute to the understanding I gained
- ○ Complicated the illustration somewhat, having a negative impact
- ○ Detracted from the understanding I gained
- ○ I did not complete this activity

* **The 'using variables' activity is intended to illustrate how a program is can store and manipulate information. Do you think this activity:**
Choose one of the following answers

- ○ Illustrated the concept well and I am confident that I understand the concept
- ○ Illustrated the concept well but I am not confident that I understand the concept fully
- ○ Illustrated the concept to some degree and this helped me somewhat to understand the concept
- ○ Illustrated the concept to some degree but this did not help my understanding
- ○ Did not really illustrate the concept
- ○ I did not complete this activity

* **The graphical nature of the 'using variables' activity: Choose one of the following answers**

- ○ Was central to the illustration and the understanding to be gained from the activity
- ○ Contributed in a positive fashion to the illustration and my understanding
- ○ Helped in my gaining an understanding
- ○ Did not really contribute to the understanding I gained
- ○ Complicated the illustration somewhat, having a negative impact
- ○ Detracted from the understanding I gained
- ○ I did not complete this activity

* **The 'using a conditional' activity illustrated 'choice making' or 'branching' in a program. Do you think this activity: Choose one of the following answers**

- ○ Illustrated the concept well and I am confident that I understand the concept
- ○ Illustrated the concept well but I am not confident that I understand the concept fully
- ○ Illustrated the concept to some degree and this helped me somewhat to understand the concept
- ○ Illustrated the concept to some degree but this did not help my understanding
- ○ Did not really illustrate the concept
- ○ I did not complete this activity

The graphical nature of the 'using a conditional' activity: Choose one of the following answers

- ○ Was central to the illustration and the understanding to be gained from the activity
- ○ Contributed in a positive fashion to the illustration and my understanding
- ○ Helped in my gaining an understanding
- ○ Did not really contribute to the understanding I gained
- ○ Complicated the illustration somewhat, having a negative impact
- ○ Detracted from the understanding I gained
- ○ I did not complete this activity
- ◉ No answer

* **The 'using a loop' activity was intended to illustrate how steps can be performed multiple times. Do you think this activity:**
Choose one of the following answers

- ○ Illustrated the concept well and I am confident that I understand the concept
- ○ Illustrated the concept well but I am not confident that I understand the concept fully
- ○ Illustrated the concept to some degree and this helped me somewhat to understand the concept
- ○ Illustrated the concept to some degree but this did not help my understanding
- ○ Did not really illustrate the concept
- ○ I did not complete this activity

* **The graphical nature of the 'using a loop' activity:**
Choose one of the following answers

- ○ Was central to the illustration and the understanding to be gained from the activity
- ○ Contributed in a positive fashion to the illustration and my understanding
- ○ Helped in my gaining an understanding
- ○ Did not really contribute to the understanding I gained
- ○ Complicated the illustration somewhat, having a negative impact
- ○ Detracted from the understanding I gained
- ○ I did not complete this activity

Understanding textual programming constructs

These two questions ask you about the help that the graphical activities provided for your understanding of the example textual conditional and loop constructs.

* **When you came to examine the textual loop statement, did you find:**
Choose one of the following answers

- ○ it to be a straightforward matter to understand the textual version of the statement based on your understanding of the graphical illustration
- ○ the textual version of the statement straightforward to understand and the graphical illustration aided you in this
- ○ the textual version of the statement straightforward to understand but the graphical illustration of the statement did not help you
- ○ the textual version of the statement straightforward to understand but did not identify any similarity to the graphical illustration
- ○ the textual version of the statement straightforward and easier to understand than the graphical illustration
- ○ the textual version of the statement difficult to understand but the graphical illustration helped
- ○ the textual version of the statement difficult to understand and the graphical illustration did not help

* **When you came to examine the textual conditional statement, did you find:**
Choose one of the following answers

- ○ it to be a straightforward matter to understand the textual version of the statement based on your understanding of the graphical illustration
- ○ the textual version of the statement straightforward to understand and the graphical illustration aided you in this
- ○ the textual version of the statement straightforward to understand but the graphical illustration of the statement did not help you
- ○ the textual version of the statement straightforward to understand but did not identify any similarity to the graphical illustration
- ○ the textual version of the statement straightforward and easier to understand than the graphical illustration
- ○ the textual version of the statement difficult to understand but the graphical illustration helped
- ○ the textual version of the statement difficult to understand and the graphical illustration did not help

Using Scratch

The next group of questions ask you about your experience with the Scratch software and graphical programming language.

* **Overall did you find the Scratch tool:**
Check any that apply

- ☐ Simple to install
- ☐ Simple to get started with programming
- ☐ Easy to use for programming
- ☐ Difficult to get started with
- ☐ Difficult to install
- ☐ Fun to use
- ☐ Frustrating to use
- ☐ Easy to get start but more difficult later on
- ☐ Difficult to get started with but easier later on
- ☐ Something you could not wait to stop using
- ☐ Something you will continue to use

Do you think that the Scratch graphical approach to programming is:
Check any that apply

- ☐ Is easy to understand and use
- ☐ Is easier to use than a text based programming language
- ☐ Is sufficient for my programming interests
- ☐ Is not powerful enough for my programming interests
- ☐ Is better for learning programming than a textual language
- ☐ Is not really different to a textual language
- ☐ Is just as hard to understand as a textual language
- ☐ Is not good for an advanced programmer
- ☐ Is not good for learning programming
- ☐ Encourages me to learn to programme
- ☐ Encourages me to also learn a textual language
- ☐ Is confusing or unclear in some aspects
- ☐ Is constraining in what I can do

Other Comments

Please tell us anything else relating to using Scratch and this survey that you wish. You might like to describe any specific problems or positive experiences you have had using Scratch or provide additional detail for any of the questions asked earlier.